Logistics

- Canvas groups for Labs
- Lab 1 due tomorrow
- Section tomorrow for Lab 2
    - Due on Jan 31
    - Late days allowed
- Problem set 1 will be released on Friday
    - Due on Jan 27
    - Late days not allowed

Question

Can more than one server think it is the primary at the same time?

Split brain

1:A,B

A is still up, but can’t reach view server
(or is unlucky and pings get dropped)

2:B,_

B learns it is promoted to primary
A still thinks it is primary

Split brain

Can more than one server *act* as primary?

- Act as = respond to clients

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Incomplete state

1:A,B

A is still up, but can’t reach view server

2:C,D

C learns it is promoted to primary
A still thinks it is primary
C doesn’t know previous state

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*

2. Primary must wait for backup to accept/execute each op before doing op and replying to client

3. Backup must accept forwarded requests only if view is correct

4. Non-primary must reject client requests

5. Every operation must be before or after state transfer

1. Missing writes

1:A,B

Client writes to A, receives response
A crashes before writing to B

2:B,C

Client reads from B
Write is missing

2. “Fast” Reads?

Does the primary need to forward reads to the backup?

(This is a common “optimization”)

Stale reads

1:A,B

A is still up, but can’t reach view server

2:B,C

Client 1 writes to B
Client 2 reads from A
A returns outdated value

Reads vs. writes

Reads treated as state machine operations too

But: can be executed more than once

RPC library can handle them differently

Rules

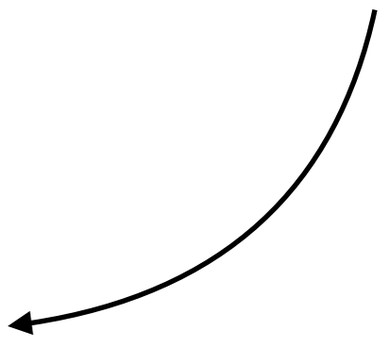
1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Partially split brain

1:A,B

2:B,C

A forwards a request…



Which arrives here

Old messages

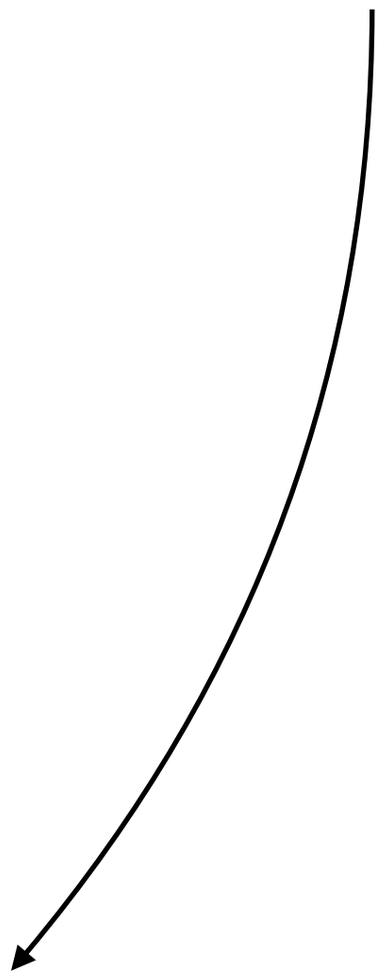
1:A,B

2:B,C

3:C,A

4:A,B

A forwards a request…



Which arrives here

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Inconsistencies

1:A,B

2:B,C

3:B,A

Outdated client sends request to A
A shouldn’t respond!

What about old messages to primary?

1:A,B

2:B,C

3:B,A

4:A,D

Outdated client sends request to A

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Inconsistencies

1:A,B

A starts sending state to B
Client writes to A
A forwards op to B
A sends rest of state to B

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Progress

Are there cases when the system can’t make further progress (i.e. process new client requests)?

Progress

- View server fails
- Network fails entirely (hard to get around this one)
- Client can’t reach primary but it can ping VS
- No backup and primary fails
- Primary fails before completing state transfer

State transfer and RPCs

State transfer must include RPC data

Duplicate writes

1:A,B

Client writes to A
A forwards to B
A replies to client
Reply is dropped

2:B,C

B transfers state to C, crashes

3:C,D

Client resends write. Duplicated!

One more corner case

1:A,B

View server stops hearing from A
A and B, and clients, can still communicate

2:B,C

B hasn’t heard from view server
Client in view 1 sends a request to A
What should happen?
Client in view 2 sends a request to B
What should happen?

Replicated Virtual Machines

Whole system replication

Completely transparent to applications and clients

High availability for any existing software

Challenge: Need state at backup to exactly mirror primary

Restricted to uniprocessor VMs

Deterministic Replay

Key idea: state of VM depends only on its input

- Content of all input/output

- Precise instruction of every interrupt

  - Only a few exceptions (e.g., timestamp instruction)

Record all hardware events into a log

- Modern processors have instruction counters and can interrupt after (precisely) x instructions

- Trap and emulate any non-deterministic instructions

Replicated Virtual Machines

Replay I/O, interrupts, etc. at the backup

- Backup executes events at primary with a lag
- Backup stalls until it knows timing of next event
- Backup does not perform external events

Primary stalls until it knows backup has copy of every event up to (and incl.) output event

- Then it is safe to perform output

Detailed Example

Primary receives network interrupt

hypervisor forwards interrupt plus data to backup

hypervisor delivers network interrupt to OS kernel

OS kernel runs, kernel delivers packet to server

server/kernel write response to network card

hypervisor delays sending response to client until backup acks

Backup receives log entries

backup delivers network interrupt

…

hypervisor does *not* put response on the wire

hypervisor ignores local clock interrupts